

# Seeded PageRank Solution Paths

D. F. Gleich<sup>1</sup>, and K. Kloster<sup>2</sup>

<sup>1</sup> *Department of Computer Science, Purdue University, West Lafayette IN, USA*  
*email: dgleich@purdue.edu*

<sup>2</sup> *Department of Mathematics, Purdue University, West Lafayette IN, USA*  
*email: kkloste@purdue.edu*

We study the behavior of network diffusions based on the PageRank random walk from a set of seed nodes. These diffusions are known to reveal small, localized clusters (or communities) and also large macro-scale clusters by varying a parameter that has a dual-interpretation as an accuracy bound and as a regularization level. We propose a new method that quickly approximates the result of the diffusion for all values of this parameter. Our method efficiently generates an approximate *solution path* or *regularization path* associated with a PageRank diffusion, and it reveals cluster structures at multiple size-scales between small and large. We formally prove a runtime bound on this method that is independent of the size of the network, and we investigate multiple optimizations to our method that can be more practical in some settings. We demonstrate that these methods identify refined clustering structure on a number of real-world networks with up to 2 billion edges.

**Key Words:** 05C81 Random walks on graphs; 05C50 Graphs and linear algebra (matrices, eigenvalues, etc.); 90C35 Programming involving graphs or networks; 91D30 Social networks; 05C82 Small world graphs, complex networks

---

## 1 Introduction

Networks describing complex technological and social systems display many types of structure. One of the most important types of structure is clustering because it reveals the modules of technological systems and communities within social systems. A tremendous number of methods and objectives have been proposed for this task (survey articles include refs. [26, 30]). The vast majority of these methods seek large regions of the graph that display evidence of local structure. For the case of modularity clustering, methods seek statistically anomalous regions; for the case of conductance clustering, methods seek dense regions that are weakly connected to the rest of the graph. All of the objective functions designed for these clustering approaches implicitly or explicitly navigate a trade-off between cluster size and the underlying clustering signal. For example, large sets tend to be more anomalous than small sets. Note that these trade-offs are essential to multi-objective optimization, and the choices in the majority of methods are natural. Nevertheless, directly optimizing the objective makes it difficult to study these structures

as they vary in size from small to large because of these implicit or explicit biases. This intermediate regime represents the meso-scale structure of the network.

In this manuscript, we seek to study structures in this meso-scale regime by analyzing the behavior of seeded graph diffusions. Seeded graph diffusions model the behavior of a quantity of “dye” that is continuously injected at a small set of vertices called the *seeds* and distributed along the edges of the graph. These seeded diffusions can reveal multi-scale features of a graph through their dynamics. The class we study can be represented in terms of a column-stochastic distribution operator  $\mathbf{P}$ :

$$\mathbf{x} = \sum_{k=0}^{\infty} \gamma_k \mathbf{P}^k \mathbf{s}$$

where  $\gamma_k$  are a set of diffusion coefficients that reflect the behavior of the dye  $k$  steps away from the seed, and  $\mathbf{s}$  is a sparse, stochastic vector representing the seed nodes. More specifically, we study the PageRank diffusions

$$\mathbf{x} = \sum_{k=0}^{\infty} (1 - \alpha) \alpha^k \mathbf{P}^k \mathbf{s}.$$

The PageRank diffusion is equivalent to the stationary distribution of a random walk that (i) with probability  $\alpha$ , follows an edge in the graph and (ii) with probability  $(1 - \alpha)$  jumps back to a seed vertex (see Section 2 more detail on this connection).

PageRank itself has been used for a broad range of applications including data mining, machine learning, biology, chemistry, and neuroscience; see our recent survey [11]. Among all the uses of PageRank, the *seeded variation* is frequently used to localize the PageRank vector within a subset of the network; this is also known as *personalized PageRank* due to its origins on the web, or *localized PageRank* because of its behavior. (We will use these terms: seeded PageRank, personalized PageRank, and localized PageRank interchangeably and use the standard acronym PPR to refer to them.) Perhaps the most important justification for this use is presented in [2], where the authors determined a relationship between seeded PageRank vectors and low-conductance sets that allowed them to create a type of graph partitioning method that does not need to see the entire graph. Their PageRank-based clustering method, called the *push method*, has been used for a number of important insights into communities in large social and information networks [17, 21].

Our focus is a novel application of this push method for meso-scale structural analysis of networks. Push, which we’ll describe formally in Section 3, depends on an accuracy parameter  $\varepsilon$ . As we vary  $\varepsilon$ , the result of the push method for approximating the PageRank diffusion reveals different structures of the network. We illustrate three PageRank vectors as we vary  $\varepsilon$  for Newman’s network science collaboration graph [25] in Figure 1. There, we see that the solution vectors for PageRank that result from push have only a few non-zeros for large values of  $\varepsilon$ . (Aside: There is a subtle inaccuracy in this statement. As we shall see shortly, we actually are describing degree normalized PageRank values. This difference does not affect the non-zero components or the intuition behind the discussion.) This is interesting because an accurate PageRank vector is mathematically non-zero everywhere in the graph. Push, with large values of  $\varepsilon$ , then produces sparse approximations to the PageRank vector. This connection is formal, and the parameter  $\varepsilon$  has a dual interpretation as a sparsity regularization parameter [12] (reviewed in Section 3.2).

The solution path or regularization path for a parameter is the set of trajectories that

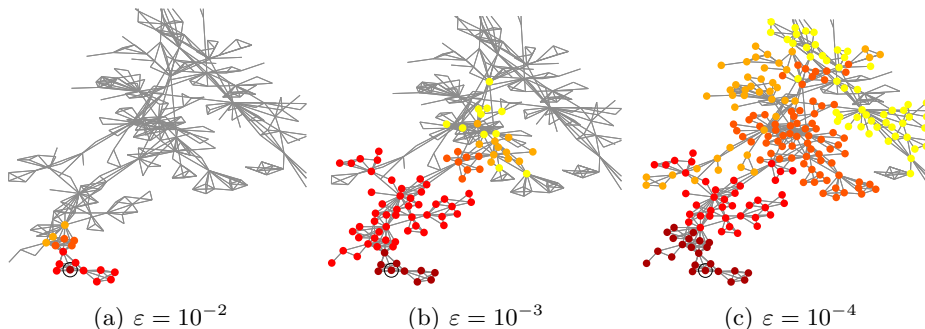


Figure 1. Nodes colored by their degree-normalized PageRank values as  $\varepsilon$  varies: dark red is large, yellow is small. The hidden nodes are mathematically zero. As  $\varepsilon$  decreases, more nodes become non-zero.

the components of the solution trace out as the parameter varies [9]. We present new algorithms based on the *push procedure* that allow us to approximate the solution path trajectories as a function of  $\varepsilon$ . We use our solution path approximation to explore the properties of graphs at many size-scales in Section 4. In our technical description, we show that the solution path remains localized in the graph (Theorem 5.1). Experiments show that it runs on real-world networks with millions of nodes in less than a second (Section 6).

The push method has become a frequently-used graph mining primitive because of the sparsity of the vectors that result from when push is used to approximate the seeded PageRank diffusion, along with the speed at which they can be computed. The method is typically used to identify sets of low-conductance in a graph as part of a community or cluster analysis [10, 13, 14, 17, 21, 28]. In these cases, the insights provided by the solution paths are unlikely to be necessary. Rather, what is needed is a faster way to compute these diffusions for many values of  $\varepsilon$ . We describe a data structure called a *shelf* that we demonstrate can use 40 times as many values of  $\varepsilon$  in only 7 times the runtime (Section 6.3).

We plan to make our computational codes available in the spirit of reproducible research.

## 2 Technical Preliminaries

We first fix our notation and review the Andersen-Chung-Lang procedure, which forms the basis for many of our contributions. We denote a graph by  $G = (V, E)$ , where  $V$  is the set of nodes and  $E$  the set of edges. All graphs we consider are simple, connected, and undirected. Let  $G$  have  $n = |V|$  nodes and fix a labeling of the graph nodes using the numbers  $1, 2, \dots, n$ . We refer to a node by its label. For each node  $j$  we denote its degree by  $d_j$ .

The *adjacency matrix* of the graph  $G$ , which we denote by  $\mathbf{A}$ , is the  $n \times n$  matrix having  $A_{i,j} = 1$  if nodes  $i$  and  $j$  are connected by an edge, and 0 otherwise. Since  $G$  is simple and undirected,  $\mathbf{A}$  is symmetric with 0s on the diagonal. The matrix  $\mathbf{D}$  denotes the diagonal matrix with entry  $(i, i)$  equal to the degree of node  $i$ ,  $d_i$ . Since  $G$  is connected,  $\mathbf{D}$  is invertible, and we can define the *random walk transition matrix*  $\mathbf{P} := \mathbf{A}\mathbf{D}^{-1}$ .

We denote by  $\mathbf{e}_j$  the standard basis vector of appropriate dimensions with a 1 in entry  $j$ , and by  $\mathbf{e}$  the vector of all 1s. In general, we use subscripts on matrices and vectors to denote entries, e.g.  $A_{i,j}$  is entry  $(i, j)$  of matrix  $\mathbf{A}$ ; the notation for standard basis vectors,  $\mathbf{e}_j$ , is an exception. Superscripts refer to vectors in a sequence of vectors, e.g.  $\mathbf{x}^{(k)}$  is the  $k$ th vector in a sequence.

For any set of nodes,  $S \subseteq V$ , we define the *volume* of  $S$  to be the sum of the degrees of the nodes in  $S$ , denoted  $\text{vol}(S) = \sum_{j \in S} d_j$ . Next, define the *boundary* of  $S \subseteq V$  to be the set of edges that have one endpoint inside  $S$  and the other endpoint outside  $S$ , denoted  $\partial(S)$ . Finally, the *conductance* of  $S$ , denoted  $\phi(S)$ , is defined by

$$\phi(S) := \frac{|\partial(S)|}{\min\{\text{vol}(S), \text{vol}(V - S)\}}.$$

Conductance can be thought of as measuring the extent to which a set is more connected to itself than the rest of the graph and is one of the most commonly used community detection objectives [26].

## 2.1 PageRank and Andersen-Chung-Lang Method

The Andersen-Chung-Lang method uses PageRank vectors to identify a set of small conductance focused around a small set of starting nodes [2]. We call such starting nodes *seed sets* and the resulting communities, *local* communities. We now briefly review this method starting with PageRank.

For a stochastic matrix  $\mathbf{P}$ , a stochastic vector  $\mathbf{v}$ , and a parameter  $\alpha \in (0, 1)$  we define the PageRank diffusion as the solution  $\mathbf{x}$  to the linear system

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}. \quad (2.1)$$

Note that when  $\alpha \in (0, 1)$  the system in (2.1) can be solved via a Neumann series expansion, and so the solution  $\mathbf{x}$  to this linear system is equivalent to the PageRank diffusion vector described in Section 1. When  $\mathbf{v} = (1/|S|)\mathbf{e}_S$ , i.e. the indicator vector for a seed set  $S$ , normalized to be stochastic, then we say the PageRank vector has been *seeded* on the set  $S$  (or *personalized* on the set  $S$ ).

Given PageRank diffusion scores  $\mathbf{x}$ , the Andersen-Chung-Lang procedure uses the values  $\mathbf{x}_j/d_j$  to determine an order for a sweep-cut procedure (described below) that identifies a set of good conductance. Thus, we would like to bound the error in approximating the values  $\mathbf{x}_j/d_j$ . Specifically (for their theory) we need our approximate solution  $\hat{\mathbf{x}}$  to satisfy

$$0 \leq \mathbf{x}_j - \hat{\mathbf{x}}_j < \varepsilon d_j \quad \text{or equivalently,} \quad \mathbf{x} \geq \hat{\mathbf{x}}, \text{ and } \|\mathbf{D}^{-1}(\mathbf{x} - \hat{\mathbf{x}})\|_\infty < \varepsilon. \quad (2.2)$$

Once a PPR diffusion  $\mathbf{x}$  is computed to this accuracy, a near-optimal conductance set located nearby the seed nodes is generated from the following *sweep cut* procedure. Rank the nodes in descending order by their scaled diffusion scores  $\mathbf{x}_j/d_j$ , with large scores ranking the highest. Denote the set of nodes ranked 1 through  $m$  by  $S(m)$ . Iteratively compute the conductance of the sets  $S(m)$  for  $m = 2, 3, \dots$ , until  $\mathbf{x}_m/d_m = 0$ . Return the set  $S(t)$  with the minimal conductance. This returned set is related to the optimal set of minimum conductance nearby the seed set through a localized Cheeger inequality [2]. The value of  $\varepsilon$  relates to the possible size of the set.

### 3 The push procedure

The push procedure is an iterative algorithm to compute a PageRank vector to satisfy the approximation (2.2). The distinguishing feature is that it can accomplish this goal with a sparse solution vector, which it can usually generate without ever looking at the entire graph or matrix. This procedure allows the Andersen-Chung-Lang procedure to run without ever looking at the entire graph. As we discussed in the introduction, this idea and method are at the heart of our contributions and so we present the method in some depth.

At each step, push updates only a single coordinate of the approximate solution like a coordinate relaxation method. We'll describe its behavior in terms of a general linear system of equations. Let  $\mathbf{M}\mathbf{x} = \mathbf{b}$  be a square linear system with 1s on the diagonal, i.e.  $M_{i,i} = 1$  for all  $i$ . Consider an iterative approximation  $\mathbf{x}^{(k)} \approx \mathbf{x}$  after  $k$  steps. The corresponding residual is  $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{M}\mathbf{x}^{(k)}$ . Let  $j$  be a row index where we want to *relax*, i.e. locally solve, the equation, and let  $r$  be the residual value there,  $r = \mathbf{r}_j^{(k)}$ . We update the solution by adding  $r$  to the corresponding entry of the solution vector,  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + r\mathbf{e}_j$ , in order to guarantee  $\mathbf{r}_j^{(k+1)} = 0$ . The residual can be efficiently updated in this case. Thus, the push method involves the operations:

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + r\mathbf{e}_j \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - r\mathbf{M}\mathbf{e}_j.\end{aligned}\tag{3.1}$$

Note that the iteration requires updating just one entry of  $\mathbf{x}^{(k)}$  and accessing only a single column of the matrix  $\mathbf{M}$ . It is this local update that enables push to solve the seeded PageRank diffusion especially efficiently.

#### 3.1 The Andersen-Chung-Lang Push Procedure for PageRank

The full algorithm for the push method applied to the PageRank linear system to compute a solution that satisfies (2.2) for a seed set  $S$  is:

1. Initialize  $\mathbf{x} = 0, \mathbf{r} = (1 - \alpha)\mathbf{e}_S$  using sparse data structures such as a hash-table.
2. Add any coordinate  $i$  of  $\mathbf{r}$  where  $\mathbf{r}_i \geq \varepsilon d_i$  to a queue  $Q$ .
3. While  $Q$  is not empty
  4. Let  $j$  be the coordinate at the front of the queue and pop this element.
  5. Set  $\mathbf{x}_j = \mathbf{x}_j + \mathbf{r}_j$
  6. Set  $\delta = \alpha \mathbf{r}_j / d_j$
  7. Set  $\mathbf{r}_j = 0$
  8. For all neighbors  $u$  of node  $j$
  9. Set  $\mathbf{r}_u = \mathbf{r}_u + \delta$
  10. If  $\mathbf{r}_u$  exceeds  $\varepsilon d_u$  after this change, add  $u$  to  $Q$ .

The queue maintains a list of all coordinates (or nodes) where the residual is larger than  $\varepsilon d_j$ . We choose coordinates to relax from this queue. Then we execute the push procedure to update the solution and residual. The residual update operates on only the nodes that neighbor the updated coordinate  $j$ . Once elements in the residual exceed the threshold, they are entered into the queue. We present the convergence theory for this method in the description of our new algorithms (Section 5).

We have presented the push method so far from a linear solver perspective. To instead view the method from a graph diffusion perspective, think of the solution vector as tracking where “dye” has concentrated in the graph and the residual as tracking where “dye” is still spreading. At each step of the method, we find a node with a sufficiently large amount of dye left (Step 4), concentrate it at that node (Step 5), then update the amount of dye that is left in the system as a result of concentrating this quantity of dye (Lines 6-10). The name *push* comes from the pattern of concentrating dye and *pushing* newly unprocessed dye to the adjacent residual entries.

Note that the value of  $\varepsilon$  plays a critical role in this method as it determines the entries that enter the queue. When  $\varepsilon$  is large, only a small number of coordinates or nodes will ever enter the queue. This will result in a sparse solution. As  $\varepsilon \rightarrow 0$ , there will be substantially more entries that enter the queue.

### 3.2 Implicit regularization from Push

To understand the sparsity that results from the push method, we introduce a slight variation on the standard push procedure. Rather than using the full update  $\mathbf{x}_j + \mathbf{r}_j$  and pushing  $\alpha \mathbf{r}_j / d_j$  to the adjacent residuals, we consider a method that takes a partial update. The form we assume is that we will leave  $\varepsilon d_j \rho$  “dye” remaining at node  $j$ . For  $\rho = 0$ , this correspond to the push procedure described above. For  $\rho = 1$ , this update will remove node  $j$  from the queue, but push as little mass as possible to the adjacent nodes such that the dye at node  $j$  will remain below  $\varepsilon d_j$ . The change is just at steps 5-7:

- 5'. Set  $\mathbf{x}_j = \mathbf{x}_j + (\mathbf{r}_j - \varepsilon d_j \rho)$
- 6'. Set  $\delta = \alpha(\mathbf{r}_j - \varepsilon d_j \rho) / d_j$
- 7'. Set  $\mathbf{r}_j = \varepsilon d_j \rho$

In previous work [12, Theorem 3], we showed that  $\rho = 1$  produces a solution vector  $\mathbf{x}$  that exactly solves a related 1-norm regularized optimization problem. The form of the problem that  $\mathbf{x}$  solves is most cleanly stated as a quadratic optimization problem in  $\mathbf{z}$ , a degree-based rescaling of the solution variable  $\mathbf{x}$ :

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \mathbf{z}^T \mathbf{Q} \mathbf{z} - \mathbf{z}^T \mathbf{g} + C \varepsilon \|\mathbf{D} \mathbf{z}\|_1 \\ & \text{subject to} && \mathbf{z} \geq 0 \end{aligned} \tag{3.2}$$

The terms of the normalization  $\mathbf{x}$  vs.  $\mathbf{z}$  and the equivalence  $\mathbf{Q}, \mathbf{g}, C$  are tedious to state exactly and uninformative to our purposes in this work. The important point is that  $\varepsilon$  can also be interpreted as a regularization parameter that governs the sparsity of the solution vector  $\mathbf{x}$ . Large values of  $\varepsilon$  increase the magnitude of the 1-norm regularizer and thus cause the solutions to be sparser. Moreover, the resulting solutions are unique as the above problem is strongly convex.

In this work, we seek algorithms to compute the solution paths or regularization paths that result from trying to use all values of  $\varepsilon$  to fully study the behavior of the diffusion. In the next section we explore some potential utilities of these paths before presenting our algorithms for computing them in Section 5.

## 4 Personalized PageRank paths

In this section we aim to show the types of insights that our solution path methodology can provide. We should remark that these are primarily designed for human interpretation. Our vision is that they would be used by an analyst that was studying a network and needed to better understand the “region” around a target node. These solution paths would then be combined with something like a graph layout framework to study these patterns in the graph. Thus, much of the analysis here will be qualitative. We demonstrate quantitative advantages to the path methodology in subsequent sections.

### 4.1 Exact paths and fast path approximations

The exact solution path for the seeded PageRank diffusion results from solving the regularized optimization problem (3.2) itself for all values of  $\varepsilon$ . This could be accomplished by using ideas similar to those used to compute solution paths for the Lasso regularizer [9]. Our algorithms and subsequent analysis evaluate approximate solution paths that result from using our push-based algorithm with  $\rho = 0.9$  (Section 5.2). In this section, we compare these approximate paths to the exact paths. We find that, while the precise numbers change, the qualitative properties are no different.

Figure 2 shows the results of such a comparison on Newman’s netscience dataset (379 nodes, 914 edges [25]). Each curve or line in the plot represents the value of a non-zero entry of an approximate PageRank vector  $\mathbf{x}_\varepsilon$  as  $\varepsilon$  varies (horizontal axis). As  $\varepsilon$  approaches 0 (and  $1/\varepsilon$  approaches  $\infty$ ), each approximate PageRank entry approaches its exact value in a monotonic manner. Alternatively, we can think of each line as the diffusion value of a node as the diffusion process spreads across the graph.

One of the plots was computed by solving for the optimality conditions of (3.2); the other plot was computed using the PPR path algorithm from Section 5.2. The values of  $\varepsilon$  are automatically determined by the algorithm itself. The plots show that for the two sets of paths have essentially identical qualitative features. For example, they reveal the same bends and inflections in individual node trajectories, as well as large gaps in PageRank values. The maximum difference between the two paths never exceeds  $1.1 \cdot 10^{-4}$ .

These results were essentially unchanged for a variety of other sample diffusions we considered, and so we decided that using  $\rho = 0.9$  was an acceptable compromise between speed and exactness. Thus, all path plots in this paper were created with  $\rho = 0.9$ , unless noted otherwise. (For analysis of the *differences* of the exact paths and  $\rho$ -paths, and in particular the behavior of the  $\rho$ -approximate paths as  $\rho$  varies, see Figure 7 below.)

### 4.2 The Seeded PageRank Solution Path Plot

We now wish to introduce a specific variation on the solution path plot that shows helpful contextual information. In the course of computation, our solution path algorithm identifies a small set of values of  $\varepsilon$  (somewhere between a few hundred to a few thousand) where it satisfies the solution criteria (5.2). At these values, we perform a sweep-cut procedure to identify the set of best conductance induced by the current solution. In the solution path plot, we display the cut-point identified by this procedure as a thick black line. All the nodes whose trajectories are above the dark black line at a particular value

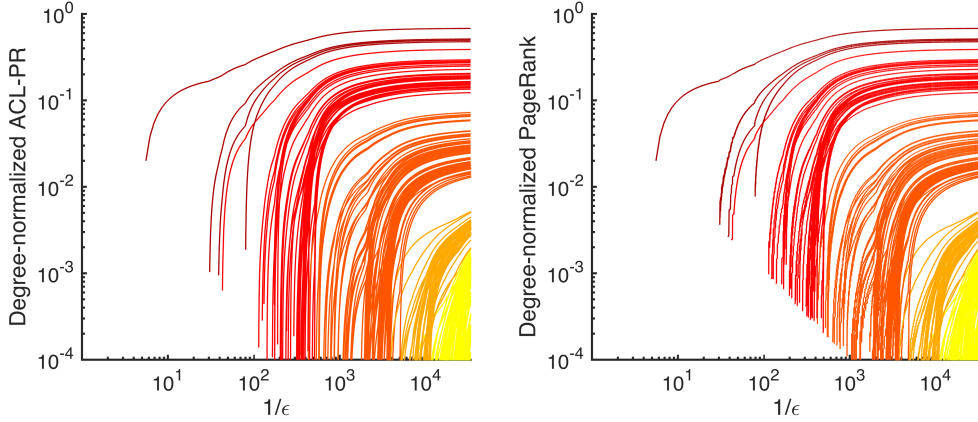


Figure 2. (Left) The solution paths for a PageRank diffusion on Newman’s netscience dataset from a single seed node computed by exactly solving the regularized problem. (Right) The approximate solution paths computed by our push-based solution path algorithm with  $\rho = 0.9$ . Each line traces a value  $\mathbf{x}_j$  as  $\varepsilon$  varies. The maximum infinity-norm distance between the two paths is  $1.1 \cdot 10^{-4}$ , showing that  $\rho = 0.9$  provides a good qualitative approximation. Moreover, the two plots highlight identical qualitative features—for example, the large gaps between paths, and the strange bend in the paths near  $\varepsilon = 10^{-3}$ . The coloring of the lines is based on the values at the smallest value of  $\varepsilon$ . The values of  $\varepsilon$  used were generated by the approximate algorithm itself and we computed the exact solution at these same values for comparison.

of  $\varepsilon$  are contained in the set of best conductance at that value of  $\varepsilon$ . This line allows us to follow the trajectory of the minimum conductance set as we vary  $\varepsilon$ . Another property of our algorithm is that the smallest possible non-zero diffusion value in the solution is  $(1 - \rho)\varepsilon$ . Thus, we plot this as a thin, diagonal, black line that acts as a pseudo-origin for all of the node trajectories. The vertical blue lines in the bottom left of the plot mark the values of  $\varepsilon$  where we detect a significant new set of best conductance. Representative conductance values are shown when there is room in the plot.

The solution path plot that corresponds to Figure 2 is shown in Figure 3. This plot illustrates all of the features we discussed in this section.

#### 4.3 Nested communities in netscience and Facebook

We now discuss some of the insights that arise from the solution path plot. In Figure 3, we show the seeded PageRank solution path plot for around 21,000 values of  $\varepsilon$  computed via our algorithm for the network science collaboration network. This computation runs in less than a second. Here, we see that large gaps in the degree normalized PageRank vector indicate cutoffs for sets of good conductance. This behavior is known to occur when sets of really good conductance emerge [1]. We can now see how they evolve and how the procedure quickly jumps between them. In particular, the path plots reveal multiple communities (good conductance sets) nested within one another through the gaps between the trajectories.



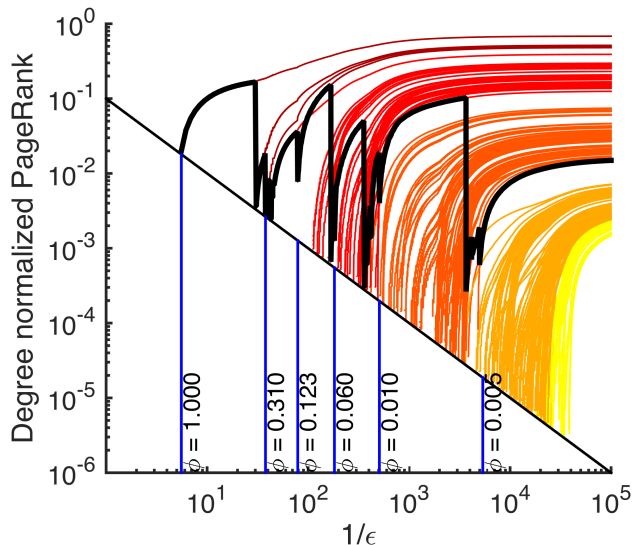


Figure 3. An example of the seeded PageRank solution path plot on Newman’s netscience dataset. Each colored line represents the value of a single node as the diffusion progresses from large  $\varepsilon$  to small  $\varepsilon$ . Because of our  $\rho$ -approximation to the true paths, the smallest value any node obtains is  $(1 - \rho)\varepsilon$  and we plot this as a dark diagonal line. The thick black line traces out the boundary of the set of best conductance found at each distinct value of  $\varepsilon$  as determined by a sweep-cut procedure. The blue lines indicate significant changes to the set of minimum conductance, and they are labelled with the conductance value. The coloring of the trajectory lines is based on the values at the smallest value of  $\varepsilon$ . We discuss implications of the plot in Section 4.3.

On a crawl of a Facebook network from 2009 where edges between nodes correspond to observed interactions [29] (see Table 1, **fb-one**, for the statistics), we are able to find a large, low conductance set using our solution path method. (Again, this takes about a second of computation.) Pictured in Figure 4, this diffusion shows no sharp drops in the PageRank values like in the network science data, yet we still find good conductance cuts. Note the few stray “orange” nodes in the sea of yellow. These nodes quickly grow in PageRank and break into the set of smallest conductance. Finding these nodes is likely to be important to understand the boundaries of communities in social networks; these trajectories could also indicate anomalous nodes. Furthermore, this example also shows evidence of multiple nested communities. These are illustrated with the manual annotations  $A, B, C$ .

#### 4.4 Core and periphery structure in the US Senate

The authors in [17] analyzed voting patterns across the first 110 US-Senates by comparing senators in particular terms. We form a graph from this US Senate data where each senator is represented by a single node. For each term of the senate, we connect senators in that session to their 3 nearest neighbors measured by voting similarities. This graph

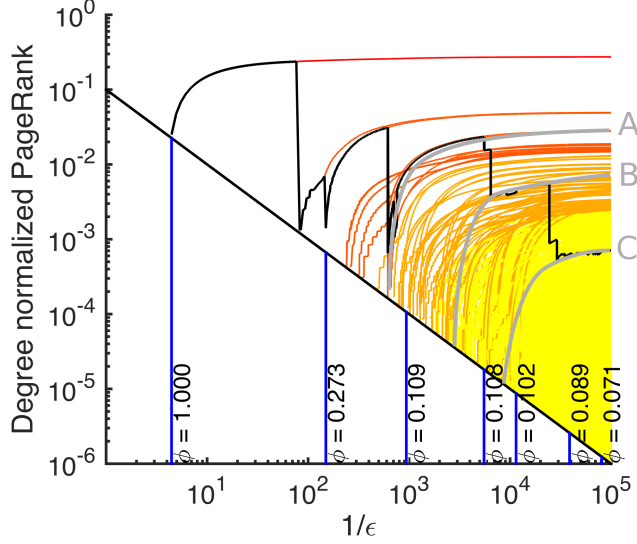


Figure 4. The seeded PageRank solution path for a crawl of observed Facebook network activity for one year (fb-one from Table 1) shows large, good cuts do not need to have large drops in the PageRank values. Nodes enter the solution and then quickly break into the best conductance set, showing that the frontier of the diffusion should be an interesting set in this graph. Furthermore, this path plot shows evidence of multiple nested communities (*A*, *B*, and *C*), which were manually annotated. The set *A* is only a few nodes, but has a small conductance score of 0.11; set *B* grows and improves this to a conductance of 0.1, and finally set *C* achieves a conductance of 0.07, which is an unusually small conductance value in a large social network.

has a substantial temporal structure as a senator from 100 years ago cannot have any direct links to a senator serving 10 years ago. We show how our solution paths display markedly different characteristics when seeded on a node near the core of the network compared with a node near the periphery. This example is especially interesting because both diffusions lead to on closely related cuts.

Figure 5 displays solution paths seeded on a senator on the periphery of the network (top right) and a senator connected to the core of the network (top left). Here are some qualitative insights from the solution path plots. The peripheral seed is a senator who served a single term; the diffusion spreads across the graph slowly because the seed is poorly connected to the network outside the seed senator’s own senate term. As the diffusion spreads outside the seed’s particular term, the paths identify multiple nested communities that essentially reflect previous and successive terms of the Senate. In contrast, the core node is a senator who served eight terms. The core node’s paths skip over such smaller-scale community structures (i.e. individual senate terms) as the diffusion spreads to each of those terms nearly simultaneously. Instead, the paths of the core node identify only one good cut: the cut separating all of the seed’s terms from the remainder of the network.

This example demonstrates the paths’ potential ability to shed light on a seed’s

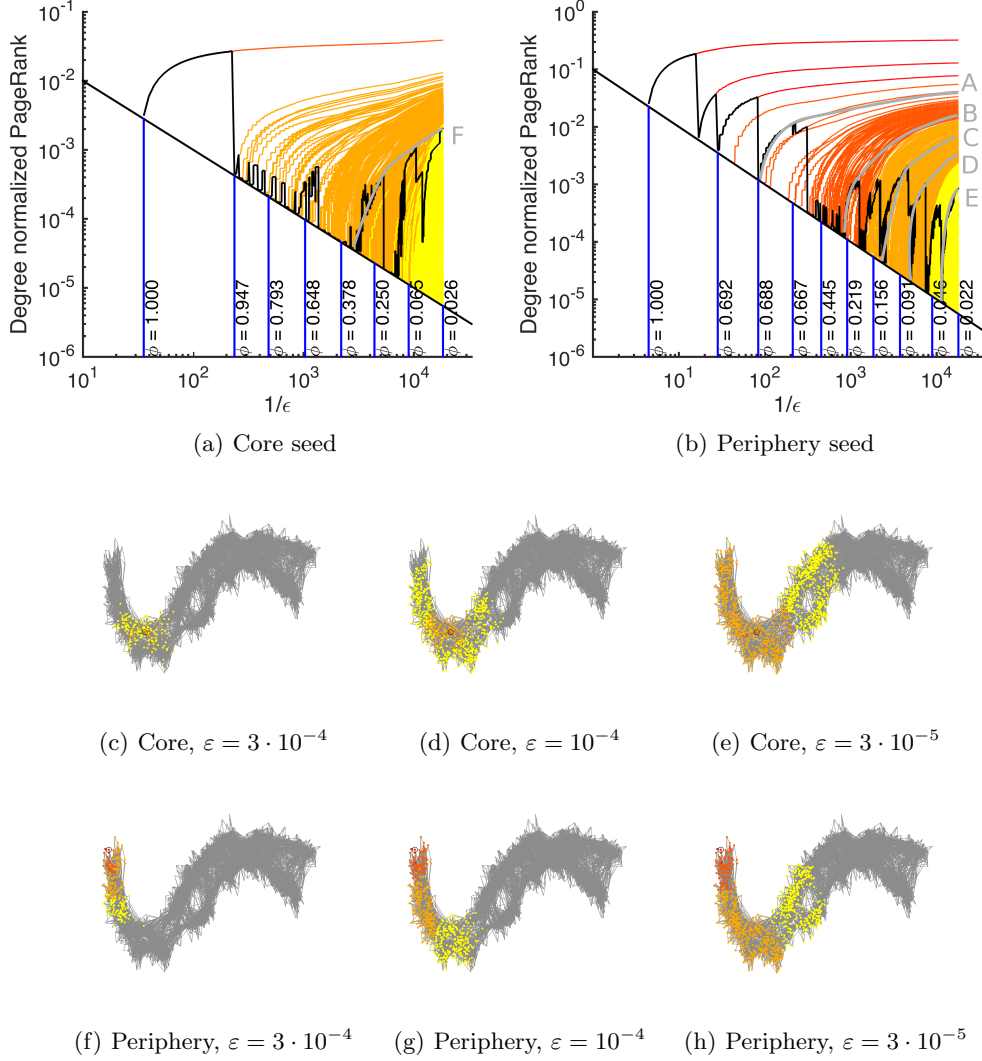


Figure 5. (*Top.*) The solution paths on the US-Senate graph for a senator in the core (who served multiple terms and is centrally located in a graph layout) and for a senator in the periphery (who served a single term and is located on the boundary of the graph layout). (*Bottom.*) The diffusions for each of these senators are shown as heat-plots on the graph layout. Red indicates nodes with the largest values and yellow the smallest. The seed nodes are circled in these layouts. The solution paths for a peripheral node indicate multiple nested communities, visible in the images of the diffusion on the whole graph and marked A, B, C, D, E. These sets are strongly correlated with successive terms of the Senate. In contrast, the core node diffusion only indicates one good cut. For the core node, we can see the diffusion essentially spreads across multiple dense regions simultaneously, without settling in one easily separated region until  $\epsilon$  is small enough that the diffusion has spread to the entire left side of the graph. The sets A and F are also almost the same.

relationship to the network’s core and periphery, as well as the seed’s relationship to many communities.

#### 4.5 Cluster boundaries in handwritten digit graphs

Finally, we use the solution paths to study the behavior of a diffusion for a semi-supervised learning task. The USPS hand-written digits dataset consists of roughly 10,000 images of the digits 0 through 9 in human hand-writing [32]. Each digit appears in roughly 1,000 of the images, and each image is labelled accordingly. From this data we construct a 3-nearest-neighbors graph, and carry out our analysis as follows. Pick one digit, and select 4 seed nodes uniformly at random from the set of nodes labelled with this digit. Then compute the PageRank solution paths from these seeds. Figure 6 shows the path plots with labels (right) and without (left). In the labelled plot, the correct labels are red and the incorrect labels are green.

We can use the best conductance set determined by the PPR vector to capture a number of other nodes sharing the seeds’ label. However, this straight-forward usage of a PageRank vector results in a number of false positives. Figure 6 (right) shows that a number of nodes with incorrect labels are included in the set of best conductance (curves that are not colored red do not share the seed’s label).

Looking at the solution-paths for this PageRank vector (Figure 6, left) we can see that a number of these false positives can be identified as the erratic lighter-orange paths cutting across the red paths. Furthermore, the solution paths display earlier sets of best conductance (left of the black spikes near  $\varepsilon = 10^{-3}$ ) that would cut out almost all false positives. This demonstrates that the solution paths can be used to identify “stable” sets of best conductance that are likely to yield higher precision labeling results. Consequently, these results hint that a smaller, but more precise, set lurks inside of the set of best conductance. This information would be valuable when determining additional labels or trying to study new data that is not as well characterized as the USPS digits dataset.

#### 4.6 Discussion

Overall, these seeded PageRank solution path plots reveal information about the clusters and sets near the seeds. Some of the features we’ve seen include nested community structure and core-periphery structure. They all provide refined information about the boundary of a community containing the seed, and suggest nodes with seemingly anomalous connections to the seed. For instance, some nodes enter the diffusion early but have only a slow-growing value indicating a weak connection to the seed; other nodes are delayed in entering the diffusion but quickly grow in magnitude and end up being significant members of the cluster. Each of these features offers refined insights over the standard single-shot diffusion computation.

### 5 Algorithms

Here we present two novel algorithms for analyzing a PPR diffusion across a variety of accuracy parameter settings by computing the diffusion only a single time. Our first

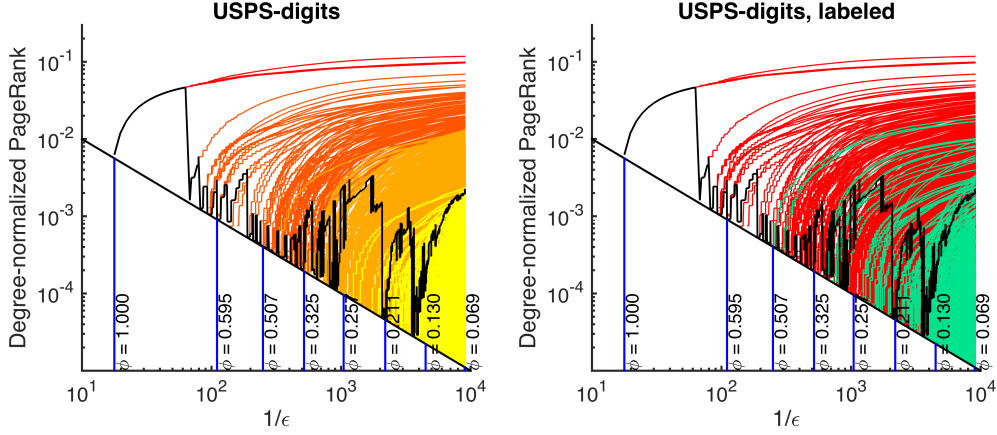


Figure 6. Seeded PageRank solution path plots for diffusions in the USPS digit dataset. The seeds are chosen to be images of handwritten digits with the same label. (*At left.*) The solution paths reveal a number of anomalous node trajectories near the set of best conductance. Nodes entering the set of best conductance after the black line erratically oscillates are most likely to be false positives near the boundary. (*At right.*) Here, we have colored the solution path lines based on the true-class label. Red shows a correct label and green shows an incorrect label.

algorithm (Section 5.2) computes the best-conductance set from the  $\rho$ -approximate solution paths described in Section 3.2. This effectively finds the best-conductance set from PPR diffusions for *every* accuracy satisfied in an interval  $[\varepsilon_{\min}, \varepsilon_{\max}]$ , where  $\varepsilon_{\min}$  and  $\varepsilon_{\max}$  are inputs. We prove the total runtime is bounded by  $O(\varepsilon_{\min}^{-2}(1-\alpha)^{-2}(1-\rho)^{-2})$ , though we believe improvements can be made to this bound. In addition to identifying the best-conductance set taken from the different approximations, the algorithm enables us to study the solution paths of PageRank, i.e. how the PPR diffusion scores change as the diffusion’s accuracy varies. Hence, we call this method **ppr-path**.

We describe a second algorithm optimized for speed (Section 5.3) in finding sets of low conductance, as the exhaustive nature of our first method generates too much intermediate data for stricter values of  $\varepsilon$ . Instead of computing the full solution paths, the second method searches for good-conductance sets over an approximate solution for each accuracy parameter taken from a grid of parameter values. The spacing of the accuracy parameters values on the grid is an additional input parameter. For this reason, we call the algorithm **ppr-grid**. For a log-spaced grid of values  $\varepsilon_0 > \varepsilon_1 > \dots > \varepsilon_N$ , we locate the best-conductance set taken from a sweep over each  $\varepsilon_k$ -approximation. The work required to compute the diffusions is bounded by  $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$ ; we show this yields a constant factor speedup over the practice of computing each diffusion separately. However, our method requires the same amount of work for performing the sweeps over each different diffusion.

We begin by describing a modification to the PageRank linear system that will simplify our notation and the exposition of our algorithm.

### 5.1 A modified PageRank linear system for the push procedure

Recall that the goal is to solve the PageRank linear system (2.1) to the accuracy condition (2.2) and then sort by the elements  $\mathbf{x}_j/d_j$ . If we multiply Equation (2.1) by  $\mathbf{D}^{-1}$ , then after some manipulation we obtain

$$(\mathbf{I} - \alpha \mathbf{P}^T) \mathbf{D}^{-1} \mathbf{x} = (1 - \alpha) \mathbf{D}^{-1} \mathbf{v}.$$

Note this transformation relies on  $\mathbf{A}$  being symmetric so that  $\mathbf{P}^T = (\mathbf{A} \mathbf{D}^{-1})^T = \mathbf{D}^{-1} \mathbf{A} = \mathbf{D}^{-1} \mathbf{P} \mathbf{D}$ . To avoid writing  $\mathbf{D}^{-1}$  repeatedly, we make the change of variables  $\mathbf{y} = (1/(1 - \alpha)) \mathbf{D}^{-1} \mathbf{x}$  and  $\mathbf{b} = \mathbf{D}^{-1} \mathbf{v}$ . The modified system is then

$$(\mathbf{I} - \alpha \mathbf{P}^T) \mathbf{y} = \mathbf{b} \quad (5.1)$$

and we set  $\mathbf{x}^{(k)} = (1 - \alpha) \mathbf{D} \mathbf{y}^{(k)}$ .

Next we use this connection between  $\mathbf{x}$  and  $\mathbf{y}$  enables us to establish a convergence criterion for our algorithms that will guarantee we obtain an approximation with the kind of accuracy typically desired for methods related to the push operation, e.g. (2.2). More concretely, to guarantee  $\|\mathbf{D}^{-1}(\mathbf{x} - \hat{\mathbf{x}})\|_\infty < \frac{1-\alpha}{\varepsilon}$ , it suffices to guarantee  $\|\mathbf{y} - \hat{\mathbf{y}}\|_\infty < \varepsilon$ , so it suffices for our purposes to bound the error of the system (5.1).

The accuracy requirement has two components: nonnegativity, and error. We relate the solution to its residual as the first step toward proving both of these. Left-multiplying the residual vector for (5.1) by  $(\mathbf{I} - \alpha \mathbf{P}^T)^{-1}$  and substituting  $\mathbf{y} = (\mathbf{I} - \alpha \mathbf{P}^T)^{-1} \mathbf{b}$ , we get

$$\mathbf{y} - \mathbf{y}^{(k)} = \left( \sum_{m=0}^{\infty} \alpha^m (\mathbf{P}^T)^m \right) \mathbf{r}^{(k)},$$

where the right-hand side replaces  $(\mathbf{I} - \alpha \mathbf{P}^T)^{-1}$  with its Neumann series. Note here that, if the right-hand side consists of all nonnegative entries, then it is guaranteed that  $\mathbf{y} - \mathbf{y}^{(k)} \geq 0$  holds. Recall from Section 3.1 that the residual update involved in the push procedure consists of adding nonnegative components to the residual, and so the residual *must* be nonnegative. Then, since  $(1 - \alpha) \mathbf{y}^{(k)} = \mathbf{D}^{-1} \mathbf{x}^{(k)}$ , this implies  $\mathbf{x} \geq \mathbf{x}^{(k)}$ , proving one component of the accuracy criteria (2.2) is satisfied.

Next we bound the error in  $\mathbf{y}$  in terms of its residual, and then control the residual's norm. Using the triangle inequality and sub-multiplicativity of the infinity norm allows us to bound  $\|\mathbf{y} - \mathbf{y}^{(k)}\|_\infty$ , which implies (2.2), with the following

$$\sum_{m=0}^{\infty} \alpha^m \left\| (\mathbf{P}^T)^m \mathbf{r}^{(k)} \right\|_\infty \leq \left( \sum_{m=0}^{\infty} \alpha^m \left\| \mathbf{P}^T \right\|_\infty^m \right) \left\| \mathbf{r}^{(k)} \right\|_\infty.$$

Finally, since  $\mathbf{P}$  is column stochastic,  $\mathbf{P}^T$  is row-stochastic, and so  $\|\mathbf{P}^T\|_\infty = 1$ . Substituting this and noting that  $\sum_{m=0}^{\infty} \alpha^m = 1/(1 - \alpha)$  allows us to bound

$$\frac{1}{1-\alpha} \left\| \mathbf{D}^{-1} \mathbf{x} - \mathbf{D}^{-1} \mathbf{x}^{(k)} \right\|_\infty = \left\| \mathbf{y} - \mathbf{y}^{(k)} \right\|_\infty \leq \frac{1}{1-\alpha} \left\| \mathbf{r}^{(k)} \right\|_\infty.$$

So to guarantee  $\mathbf{x}$  satisfies the desired accuracy, it is enough to guarantee that

$$\left\| \mathbf{r}^{(k)} \right\|_\infty < \varepsilon \quad (5.2)$$

holds, where  $\mathbf{r}^{(k)} = \mathbf{b} - (\mathbf{I} - \alpha \mathbf{P}^T) \mathbf{y}^{(k)}$  and  $\mathbf{x}^{(k)} = (1 - \alpha) \mathbf{D} \mathbf{y}^{(k)}$ . Thus, for our algorithms

to converge to the desired accuracy, it suffices to iterate until the residual norm satisfies the bound (5.2). With this terminating condition established, we can now describe our algorithm for computing the solution paths of  $\mathbf{x}_\varepsilon$  as  $\varepsilon$  varies.

## 5.2 PageRank solution paths

Recall that our goal is computing the solution paths of seeded PageRank with respect to the parameter  $\varepsilon$ . That is, we want an approximation  $\mathbf{x}_\varepsilon$  of PageRank for all  $\varepsilon$  values inside some region. Let  $\mathbf{P}$  be a stochastic matrix, choose  $\alpha$  satisfying  $0 < \alpha < 1$ , let  $\mathbf{v}$  be a stochastic vector, and set  $\mathbf{b} = \mathbf{D}^{-1}\mathbf{v}$ . Fix input parameters  $\varepsilon_{\min}$  and  $\varepsilon_{\max}$ . Then for each value  $\varepsilon_{\text{cur}} \in [\varepsilon_{\min}, \varepsilon_{\max}]$  ( $\varepsilon_{\text{cur}}$  denotes “the value of  $\varepsilon$  currently being considered”), we want an approximation  $\hat{\mathbf{y}}$  of the solution to  $(\mathbf{I} - \alpha\mathbf{P}^T)\mathbf{y} = \mathbf{b}$  that satisfies  $\|\mathbf{y} - \hat{\mathbf{y}}\|_\infty < \frac{\varepsilon_{\text{cur}}}{1-\alpha}$ . (Or rather, we want a computable approximation to this information.) As discussed in Section 3.2, we also use the approximation parameter  $\rho \in [0, 1)$  in the push step.

Given initial solution  $\mathbf{y}^{(0)} = \mathbf{0}$  and residual  $\mathbf{r}^{(0)} = \mathbf{b}$ , proceed as follows. Maintain a priority queue,  $Q(\mathbf{r})$ , of all entries of the residual that do not satisfy the convergence criterion  $r_j < \varepsilon_{\min}$ . We store the entries of  $Q(\mathbf{r})$  using a max-heap so that we can quickly determine  $\|\mathbf{r}\|_\infty$  at every step.

Each time the value  $\|\mathbf{r}\|_\infty$  reaches a new minimum, we consider the resulting solution vector to satisfy a new “current” accuracy, which we denote  $\varepsilon_{\text{cur}}$ . For each such  $\varepsilon_{\text{cur}}$  achieved, we want to perform a sweep over the solution vector. Because the sweep operation requires a sorted solution vector, we keep  $\mathbf{y}$  in a sorted array,  $L(\mathbf{y})$ . By re-sorting the solution vector each time a single entry  $\mathbf{y}_j$  is updated, we avoid having to do a full sweep for each “new”  $\varepsilon_{\text{cur}}$ -approximation. The local sorting operation is a bubblesort on a single entry; the local sweep update we describe below.

With the residual and solution vector organized in this way, we can quickly perform each step of the above iterative update. Then, iterating until  $\|\mathbf{r}\|_\infty < \varepsilon_{\min}$  guarantees convergence to the desired accuracy. Next we present the iteration in full detail.

### PPR path algorithm

The `ppr-path` algorithm performs the following iteration until the maximum entry in  $Q(\mathbf{r})$  is below the smallest parameter desired,  $\varepsilon_{\min}$ .

1. Pop the max of  $Q(\mathbf{r})$ , say entry  $j$  with value  $r$ , then set  $\mathbf{r}_j = \rho\varepsilon_{\text{cur}}$  and reheap  $Q(\mathbf{r})$ .
2. Add  $r - \rho\varepsilon_{\text{cur}}$  to  $\mathbf{y}_j$ .
3. Bubblesort entry  $\mathbf{y}_j$  in  $L(\mathbf{y})$ .
4. If  $L(\mathbf{y})$  changes, perform a local sweep update.
5. Add  $(r - \rho\varepsilon_{\text{cur}})\alpha\mathbf{P}^T\mathbf{e}_j$  to  $\mathbf{r}$ .
6. For each entry  $i$  of  $\mathbf{r}$  that was updated, if it does not satisfy  $r_i < \varepsilon_{\min}$ , then insert (or update) that entry in  $Q(\mathbf{r})$  and re-heap.
7. If  $\|\mathbf{r}\|_\infty < \varepsilon_{\text{cur}}$ , record the sweep information, then set  $\varepsilon_{\text{cur}} = \|\mathbf{r}\|_\infty$ .

When the max-heap  $Q(\mathbf{r})$  is empty, this signals that all entries of  $\mathbf{r}$  satisfy the convergence criterion  $r_j < \varepsilon_{\min}$ , and so our diffusion score approximations satisfy the accuracy requirement (2.2).

### Sweep update

The standard sweep operation over a solution vector involves sorting the entire solution vector and iteratively computing the conductance of each consecutive sweep set. Here, we re-sort the solution vector after each update by making only the local changes necessary to move entry  $\mathbf{y}_j$  to the correct ranking in  $L(\mathbf{y})$ . This is accomplished by bubblesorting the updated entry  $\mathbf{y}_j$  up the rankings in  $L(\mathbf{y})$ . Note that if  $\mathbf{y}^{(k)}$  has  $T_k$  nonzero entries, then this step can take at most  $T_k$  operations. We believe this loose upperbound can be improved. We *could* determine the new rank of node  $\mathbf{y}_j$  in work  $\log T_k$  via a binary insert. However, since we must update the rank and sweep information of each node that node  $\mathbf{y}_j$  surpasses, the asymptotic complexity would not change.

Once the node ranks have been corrected, the conductance score update proceeds as follows. Denote by  $S^{(k-1)}(m)$  the set of nodes that have rankings  $1, 2, \dots, m$  during step  $k-1$ . Assuming we have the cut-set (cut and volume) information for each of these sets, then we can update that information for the sets  $S^{(k)}(m)$  as follows.

Suppose the node that changed rankings was promoted from rank  $j$  to rank  $j - \Delta_k$ . Observe that the sets  $S^{(k)}(m)$  and their cut-set information remain the same for any set  $S^{(k)}(m)$  lying inside the rankings  $[1, \dots, j - \Delta_k - 1]$ , because the change in rankings happened entirely in the interval  $[j - \Delta_k, \dots, j]$ . This occurs for  $m < j - \Delta_k$ . Similarly, any set  $S^{(k)}(m)$  with  $m > j$  would already contain all of the nodes whose rank changed – altering the ordering within the set does not alter the conductance of that set, and so this cut-set information also need not be changed. Hence, we need to update the cut-set information for only the intermediate sets.

Now we update the cut-set information for those intermediate sets. We refer to the node that changed rank as node  $L(j)$ . Its old rank was  $j$ , and its new rank is  $j - \Delta_k$ . Note that the cut-set information for the set  $S^{(k)}(j - t)$  (for  $t = 0, \dots, \Delta_k$ ) is the exact same as that of set  $S^{(k-1)}(j - t - 1) \cup \{L(j)\}$ . In words, we introduce the node  $L(j)$  to the set  $S^{(k-1)}(j - t - 1)$  from the previous iteration, and then compute the cut-set information for the new iteration's set,  $S^{(k)}(j - t)$ , by looking at just the neighborhood of node  $L(j)$  a single time. This provides a great savings over simply reperforming the sweep procedure over the entire solution vector up to the index where the rankings changed.

If the node being operated on,  $L(j)$ , has degree  $d$ , then this process requires work  $O(d + \Delta_k)$ . As discussed above, we can upperbound  $\Delta_k$  with the total number of iterations the algorithm performs  $T_k$ .

**Theorem 5.1** *Given a random walk transition matrix  $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$ , stochastic vector  $\mathbf{v}$ , and input parameters  $\alpha \in (0, 1)$ ,  $\rho \in [0, 1)$ , and  $\varepsilon_{\max} > \varepsilon_{\min} > 0$ , our **ppr-path** algorithm outputs the best-conductance set found from sweeps over  $\varepsilon_{\text{cur}}$ -accurate degree-normalized,  $\rho$ -approximate solution vectors  $\hat{\mathbf{x}}$  to  $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}$ , for all values  $\varepsilon_{\text{cur}} \in [\varepsilon_{\min}, \varepsilon_{\max}]$ . The total work required is bounded by  $O\left(\frac{1}{\varepsilon_{\min}^2(1-\alpha)^2(1-\rho)^2}\right)$ .*

**Proof** We carry out the proof in two stages. First, we show that the basic iterative update converges in work  $O(\varepsilon_{\min}^{-1}(1 - \alpha)^{-1}(1 - \rho)^{-1})$ . Then, we show that the additional work of sorting the solution vector and sweeping is bounded by  $O(\varepsilon_{\min}^{-2}(1 - \alpha)^{-2}(1 - \rho)^{-2})$ .



**Push work.** We count the work on just the residual  $\mathbf{r}^{(k)}$  and solution vector  $\mathbf{y}^{(k)}$ . The work required to maintain the heap  $Q$  and sorted array  $L$  is accounted for below.

Each step, the push operation acts on a single entry in the residual that satisfies  $r_j \geq \varepsilon_{\min}$ . The step consists of a constant number of operations to update the residual and solution vectors (namely, updating a single entry in each). The actual amount that is removed from the residual node is  $(r_j - \rho\varepsilon_{\min})$ ; then we add  $(r_j - \rho\varepsilon_{\min})$  to the appropriate entry of the solution, and  $(r_j - \rho\varepsilon_{\min})\alpha/d_j$  to  $\mathbf{r}_i^{(k)}$  for each neighbor  $i$  of node  $j$ . Since  $j$  has  $d_j$  such neighbors, the total work in one step is bounded by  $O(d_j)$ . If  $T$  steps of the push operation are performed, then the amount of work required to obtain an accuracy of  $\varepsilon_{\min}$  is bounded by  $\sum_{t=0}^T d_{j(t)}$ , where  $j = j(t)$  is the index of the residual operated on in step  $t$ ,  $\mathbf{r}_j^{(t)}$ .

Next we bound this expression for the work done in these “push” steps. Since all entries of the solution and residual vectors are nonnegative at all times, the sum of the values  $(r_t - \rho\varepsilon_{\min})$  pushed at each step exactly equals the sum of the values  $\mathbf{y}^{(k)}$ , i.e.  $\sum_{t=0}^T (r_t - \rho\varepsilon_{\min}) = \mathbf{e}^T \mathbf{y}^{(k)}$ . Since  $\mathbf{y}^{(k)} = (1/(1-\alpha))\mathbf{D}^{-1}\mathbf{x}^{(k)}$ , we then have that the sum of entries in  $(1/(1-\alpha))\mathbf{x}^{(k)}$  equals the sum of values pushed from the residual scaled by degree and  $(1-\alpha)$ , i.e.  $\mathbf{e}^T \mathbf{x}^{(k)} = (1-\alpha) \sum_{t=0}^T (r_t - \rho\varepsilon_{\min}) \cdot d_{j(t)}$ , where  $j(t)$  is the node pushed in step  $t$ . We claim that the sum  $\mathbf{e}^T \mathbf{x}^{(k)} \leq 1$ . Assuming this for the moment, we get from the previous equation that  $(1-\alpha) \sum_{t=0}^T (r_t - \rho\varepsilon_{\min}) \cdot d_{j(t)} = \mathbf{e}^T \mathbf{x}^{(k)} \leq 1$ . Since each step of **ppr-path** operates on a residual value satisfying  $r_t \geq \varepsilon_{\min}$ , we know that  $(r_t - \rho\varepsilon_{\min}) \geq \varepsilon_{\min}(1-\rho)$ , and so

$$(1-\alpha) \sum_{t=0}^T \varepsilon_{\min}(1-\rho) \cdot d_{j(t)} < (1-\alpha) \sum_{t=0}^T r_t \cdot d_{j(t)} \leq 1.$$

Dividing by  $\varepsilon_{\min}(1-\alpha)(1-\rho)$  completes the proof that the expression for work,  $\sum_{t=0}^T d_{j(t)}$ , is bounded by  $O(\varepsilon_{\min}^{-1}(1-\alpha)^{-1}(1-\rho)^{-1})$ .

Lastly, we justify the claim  $\mathbf{e}^T \mathbf{x}^{(k)} \leq 1$ . Left-multiplying the equations in (5.1) by  $(\mathbf{D}\mathbf{e})^T$  and using stochasticity of  $\mathbf{v}$  gives

$$\begin{aligned} \mathbf{e}^T(\mathbf{I} - \alpha\mathbf{P})\mathbf{D}\mathbf{y}^{(k)} &= \mathbf{e}^T\mathbf{D}\mathbf{b} - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)} \\ (1-\alpha)\mathbf{e}^T \frac{1}{(1-\alpha)}\mathbf{x}^{(k)} &= \mathbf{e}^T\mathbf{v} - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)} \\ \mathbf{e}^T \mathbf{x}^{(k)} &= 1 - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)}. \end{aligned} \tag{5.3}$$

As noted above, all entries of the residual and iterative solution vector are nonnegative at all times. The sum  $\mathbf{e}^T \mathbf{x}^{(k)}$  cannot exceed 1, then, because that would imply that the residual summed to a negative number, contradicting nonnegativity of the residual vector. Hence,  $\mathbf{e}^T \mathbf{x}^{(k)} \leq 1$ .

**Sorting and sweeping work.** Here we account for the work performed each step in maintaining the residual heap  $Q(\mathbf{r})$ , re-sorting the solution vector  $L(\mathbf{y})$ , and updating the sweep information for  $L(\mathbf{y})$ . To ease the process, we first fix some notation: denote the number of entries in the residual heap  $Q(\mathbf{r})$  by  $|Q|$ , and the number of non-zero entries in the sorted solution vector  $L(\mathbf{y})$  by  $|L|$ . We will bound both of these quantities later on. We continue to use  $\Delta_t$  to denote the number of rank positions changed in  $L(\mathbf{y})$  in

step  $t$ . Finally, recall that  $T$  denotes the number of iterations of the algorithm required to terminate.

The work bounds we will prove, listed in the order in which the **ppr-path** algorithm performs them, are as follows:

Operation	actual work	upperbound
Find $\max(\mathbf{r})$	1	1
Delete $\max(\mathbf{r})$	$\log( Q )$	$\log(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)})$
Bubblesort $L(\mathbf{y}_j)$	$\Delta_t$	$T$
Re-sweep $L(\mathbf{y})$	$d_j + \Delta_t$	$d_j + T$
Update $\mathbf{r} + r\alpha\mathbf{P}^T\mathbf{e}_j$	$d_j$	$d_j$
Re-heap $Q(\mathbf{r})$	$d_j \log( Q )$	$d_j \log(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)})$

The residual heap operations for deleting  $\max Q(\mathbf{r})$  and re-heapings the updated entries each require  $O(\log(|Q|))$  work, where  $|Q|$  is the size of the heap, i.e. the number of nonzero entries in the residual. We can upperbound this number using the total number of pushes performed (since a nonzero in the residual can exist only via a push operation placing it there). We bound  $|Q|$  by  $O(\varepsilon_{\min}^{-1}(1-\alpha)^{-1}(1-\rho)^{-1})$ , then. We remark that this is quite loose, as values of  $\rho$  near 1 actually force the solution and residual to be *sparser*, so the heap size should still be bounded by  $O(\varepsilon_{\min}^{-1}(1-\alpha)^{-1})$ , though we do not yet have a proof of this.

Re-sorting the solution vector via a bubblesort can involve no more operations than the length of the solution vector. Since a nonzero in entry  $\mathbf{y}_j$  can exist only if a step of the algorithm operates on an entry  $\mathbf{r}_j$ , the number of nonzeros in  $\mathbf{y}$  is bounded by the number of steps of the algorithm, i.e.  $|L| \leq T$ . We believe this bound to be loose, but cannot currently tighten it. Note that the work required in updating sweep information also requires  $\Delta_t$  work, which we again upperbound by  $T$ . The  $d_j$  term in updating sweep information is from accessing the neighbors of the entry  $\mathbf{y}_j$ , the node changing its rank.

The dominant terms in the above expression for work are the re-heap updates and the bubblesort and re-sweep operations, which require a total of  $O(d_j \log(|Q|) + |L|)$  work each step. Summing this over all  $T$  steps of the algorithm, we can majorize work by  $O(\log(|Q|) \cdot \sum_{t=0}^T d_j) + O(\sum_{t=0}^T |L|)$ , which is upperbounded by  $O\left(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)} \log(|Q|) + T \cdot |L|\right)$ . Finally, substituting in our loose upperbounds for  $T$ ,  $|Q|$ , and  $|L|$  mentioned above completes the proof:

$$O\left(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)} \log\left(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)}\right) + \frac{1}{\varepsilon_{\min}^2(1-\alpha)^2(1-\rho)^2}\right) \leq O\left(\frac{1}{\varepsilon_{\min}^2(1-\alpha)^2(1-\rho)^2}\right).$$

□

### 5.3 Fast multi-parameter PPR

Here we present a fast framework for computing  $\varepsilon$ -approximations of a push-based PPR diffusion without computing a new diffusion for each  $\varepsilon$ . This enables us to identify the optimal output that would result from multiple diffusion computations for different  $\varepsilon$  values, but without having to do the work of computing a new diffusion for each different  $\varepsilon$ . This algorithmic framework does not admit the parameter  $\rho$  as easily, because

of implementation details surrounding the data structures used to handle sorting and updating the residual.

The framework is compatible with every set of parameter choices for  $\varepsilon$  that allows for constant-time bin look-ups. More precisely, the set of parameters  $\varepsilon_0, \varepsilon_1, \dots, \varepsilon_N$  must have an efficient method for determining the index  $k$  such that, given a value  $r$ , we have  $\varepsilon_{k-1} > r \geq \varepsilon_k$ . We focus on a set of  $\varepsilon$  values that are taken from a log-spaced grid: that is, the parameters are of the form  $\varepsilon_k = \varepsilon_0 \theta^k$  for constants  $0 < \varepsilon_0, \theta < 1$ . Because we assume our  $\varepsilon$  parameters are taken from such a grid, we call our method **ppr-grid**. Another possibly useful case is choosing  $\varepsilon_k$  values taken from a grid formed from Chebyshev-like nodes, allowing for constant-time shelf-placement via  $\cos^{-1}$  evaluations.

We emphasize that the underlying algorithm we use to compute the PageRank diffusion is closely related to the push method discussed in Section 3 as implemented by [2]; in the case that only a single accuracy parameter is used, the algorithms are identical. When more than one accuracy setting is used, we employ a special data structure, which we call a shelf.

### The shelf structure

The main difference between our algorithm **ppr-grid** and previous implementations of the push method lies in our data structure replacing the priority queue,  $Q$ , discussed in **ppr-path**. Instead of inserting residual entries in a heap as in **ppr-path**, we organize them in a system of arrays. Each array holds entries between consecutive values of  $\varepsilon_k$ , so that each array holds entries larger than the shelf below it. For this reason, we call this system of arrays a “max-shelf”,  $H$ , and refer to each individual array as a “shelf”,  $H_k$ .

The process is effectively a bucket sort: each shelf (or bucket) of  $H$  holds entries of the residual lying between consecutive values of  $\varepsilon_k$  in the parameter grid. For parameters  $\varepsilon_0, \varepsilon_1, \dots, \varepsilon_N$ , shelf  $H_k$  holds residual values  $r$  satisfying  $\varepsilon_{k-1} > r \geq \varepsilon_k$ . Residual entries smaller than  $\varepsilon_N$  are omitted from  $H$  (since convergence does not require operating on them). Residual entries with values greater than  $\varepsilon_0$  are simply placed in shelf  $H_0$ .

### PPR on a grid of $\varepsilon$ parameters

During the iterative step of **ppr-grid**, then, rather than place a residual entry at the back of  $Q$ , we instead place the entry at the back of the appropriate shelf,  $H_k$ . Once all shelves  $H_m(\mathbf{r})$  are cleared for  $m \leq k$ , then the residual has no entries larger than  $\varepsilon_k$ , and so we have arrived at an approximation vector satisfying convergence criterion (2.2) with accuracy  $\varepsilon_k$ . At this point, we perform a sweep procedure using the  $\varepsilon_k$ -solution. We then repeat the process until the next shelf is cleared, and a new  $\varepsilon_{k+1}$ -solution is produced.

**PPR grid algorithm.** The iterative step is as follows:

1. Determine the top-most non-empty shelf,  $H_k$ .
2. While  $H$  contains an entry in shelf  $k$  or above, do the following:
  3. Pop an entry on or above shelf  $H_k$ , say value  $r$  in entry  $\mathbf{r}_j$ , and set  $\mathbf{r}_j = 0$ .
  4. Add  $r$  to  $\mathbf{x}_j$ .
  5. Add  $r\alpha\mathbf{P}^T\mathbf{e}_j$  to  $\mathbf{r}$ .
  6. For each entry of  $\mathbf{r}$  that was updated, move that node to the correct shelf,  $H_m$ ,

where  $\varepsilon_{m-1} > r \geq \varepsilon_m$ . If an entry is placed on a shelf higher than  $k$ , record the new top-shelf.

7. Shelves 0 through  $k$  are cleared, so the  $\varepsilon_k$ -solution is done; perform a sweep.

Once all shelves are empty, the approximation with strictest accuracy,  $\varepsilon_N$ , has been attained, and a final sweep procedure is performed.

**Shelf computation.** In each iteration of **ppr-grid** we must place multiple entries into their respective “shelves”. Here we show that computing the correct shelf where a value  $r$  will be placed can be accomplished in constant time.

Let  $\varepsilon_k = \varepsilon_0 \theta^k$  for a fixed value of  $\theta \in (0, 1)$ . We want a value  $r$  satisfying  $\varepsilon_{k-1} > r \geq \varepsilon_k$  to be placed on shelf  $k$ . If  $r \geq \varepsilon_0$ , then we place  $r$  into shelf 0. Otherwise, making the substitution  $\varepsilon_k = \varepsilon_0 \theta^k$  and performing some algebra yields

$$k - 1 < \frac{\log(r/\varepsilon_0)}{\log(\theta)} \leq k,$$

so  $k$  can be computed by taking the ceiling of  $\log(r/\varepsilon_0)/\log(\theta)$ , which is a constant time operation. Note that this process requires that  $0 < \varepsilon_k < 1$  holds for all  $k$ , that  $\theta \in (0, 1)$ , and that  $r > 0$ .

**Top shelf.** Each step of **ppr-grid** also requires determining the top non-empty shelf. This can be done in constant time by tracking what the top shelf is during each residual update. If  $k$  is the top shelf immediately prior to step (2.4), then  $k$  will still be the top shelf after the residual update is complete, unless one of the updates in step (6.) moves an entry to a shelf  $l < k$ . By checking for this event during the update of each individual residual entry in step (6.), we will have knowledge of the top non-empty shelf at the beginning of each step, with only constant work per step.

Once the current working shelf is emptied, then it is possible that the next non-empty shelf is many shelves down, i.e. shelves  $H_k$  and higher are emptied and the next non-empty shelf is  $H_{k+c}$  for some large number  $c$ . Then determining  $k + c$  takes  $O(c)$  operations. However, this operation is performed every time the algorithm switches from one value of  $\varepsilon_k$  to the next. If there are  $N$  values of  $\varepsilon_k$ , then the total work in all calls of this top-shelf computation is bounded by  $O(N)$ .

**Theorem 5.2** *Given a random walk transition matrix  $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$ , stochastic vector  $\mathbf{v}$ , and input parameters  $\alpha, \theta \in (0, 1)$  and  $\varepsilon_k = \varepsilon_0 \theta^k$ , our **ppr-grid** algorithm outputs the best-conductance set found from sweeps over  $\varepsilon_k$ -accurate degree-normalized solution vectors  $\hat{\mathbf{x}}$  to  $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}$ , for all values  $\varepsilon_k$  for  $k = 0$  through  $N$ . The work in computing the diffusions is bounded by  $O(\frac{1}{\varepsilon_N(1-\alpha)})$ . This improves on the method of computing the  $N$  diffusions separately, which is bounded by  $O(\frac{1}{\varepsilon_N(1-\alpha)(1-\theta)}(1 - \theta^{N+1}))$ . The two methods perform the same amount of sweep-cut work.*

**Proof.** Note that the amount of push-work required to produce a diffusion with smallest accuracy  $\varepsilon_N$  is exactly the same as the push-work performed in computing an  $\varepsilon_N$  solution via **ppr-path**; The only difference is in how we organize the residual and solution vectors. Hence, the push-work for **ppr-grid** is bounded by  $O(\varepsilon_N^{-1}(1 - \alpha)^{-1})$ . Updating the shelf structure for **ppr-grid** requires only a constant number of operations in each iteration,

Table 1. Datasets

Graph	$ V $	$ E $	$d_{\text{ave}}$
itdk0304	190,914	607,610	6.37
dblp	226,413	716,460	6.33
youtube	1,134,890	2,987,624	5.27
fb-one	1,138,557	4,404,989	3.9
fbA	3,097,165	23,667,394	15.3
ljournal	5,363,260	49,514,271	18.5
hollywood	1,139,905	56,375,711	98.9
twitter	41,652,230	2,041,892,992	98
friendster	65,608,366	1,806,067,135	55.1

and so the dominating operation in one step of **ppr-grid** is the residual push work. Thus, the push-work bound for **ppr-grid** is  $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$ .

**Push-work for  $N$  separate diffusions.** As noted above, computing a diffusion with parameters  $\varepsilon_k$  and  $\alpha$  requires push-work  $O(\varepsilon_k^{-1}(1-\alpha)^{-1})$ . Summing this over all values of  $\varepsilon_k$  gives  $\sum_{k=0}^N \varepsilon_k^{-1}(1-\alpha)^{-1} = (1-\alpha)^{-1} \sum_{k=0}^N (1/\varepsilon_k)$ . Substituting  $\varepsilon_0 \theta^k$  in place of  $\varepsilon_k$ , we see this sum is simply a scaled partial geometric series,  $\sum_{k=0}^N \varepsilon_k^{-1} = \varepsilon_0^{-1} \theta^{-N} (1 - \theta^{N+1}) / (1 - \theta)$ . Simplifying gives

$$\sum_{k=0}^N \frac{1}{\varepsilon_k(1-\alpha)} = \frac{1}{\varepsilon_N(1-\alpha)(1-\theta)} (1 - \theta^{N+1}),$$

proving the bound on the push-work. For our choices  $\varepsilon_0 = 10^{-1}$ ,  $\varepsilon_N = 10^{-6}/3$ , and  $\theta = 0.66$  (which corresponds to using  $N = 32$  diffusions), this quantity is roughly 2.9 times greater than computing only one diffusion, as our method does.

**Sweep work.** The number of operations required in computing the diffusion is bounded by  $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$ , but this does not include the work done in sweeping over the various  $\varepsilon_k$ -approximation vectors. The sweep operation requires sorting the solution vector. As noted in the proof of work for **ppr-path**, the number of nonzeros in the solution vector is bounded by  $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$ , and so the sorting work is  $O(\varepsilon_N^{-1}(1-\alpha)^{-1} \log(\varepsilon_N^{-1}(1-\alpha)^{-1}))$ . This implies that sorting is the dominant subroutine of the algorithm. In practice the bound on the number of nonzeros in the solution is loose, and the push operations comprise most of the labor.

## 6 Experimental Results on Finding Small Conductance Sets

We have presented two frameworks for computing a single personalized PageRank diffusion across multiple parameter settings. Here we analyze their performance on a set of real-world social and information networks with varying sizes and edge-densities with the goal of identifying sets of small conductance. All datasets were altered to be symmetric and have 0s on their diagonals; this is done by deleting any self-edges and making all directed edges undirected. In addition to versions of the Facebook dataset analyzed in Section 4, we test our algorithms on graphs including twitter-2010 from [19], friendster and youtube from [24, 31], dblp-2010 and hollywood-2009 in [3, 4], idk0304 from [27], and ljournal-2008 in [7]. See Table 1 for a summary of their properties.

### 6.1 The effect of $\rho$ on conductance

Our first experimental study regards the selection of the parameter  $\rho$  for finding sets of small conductance. We already established that  $\rho = 0.9$  yielded qualitatively accurate solution path plots. However, for the specific problem of identifying small conductance sets, we find a curious behavior and get the best results with small values of  $\rho$ . We'll explain why this is shortly, but consider the results in Figure 7. In the left subplot, we see the maximum difference between the minimum conductance found for any value of  $\rho$  over a series of trials. It can be large, for instance, 0.7 for one trial on the LiveJournal graph, where large  $\rho$  shows worse results. In that same figure, we show the runtime scaling. It seems to scale with  $1/(1 - \rho)$ , which is slightly better than expected from the bound in Theorem 5.1.

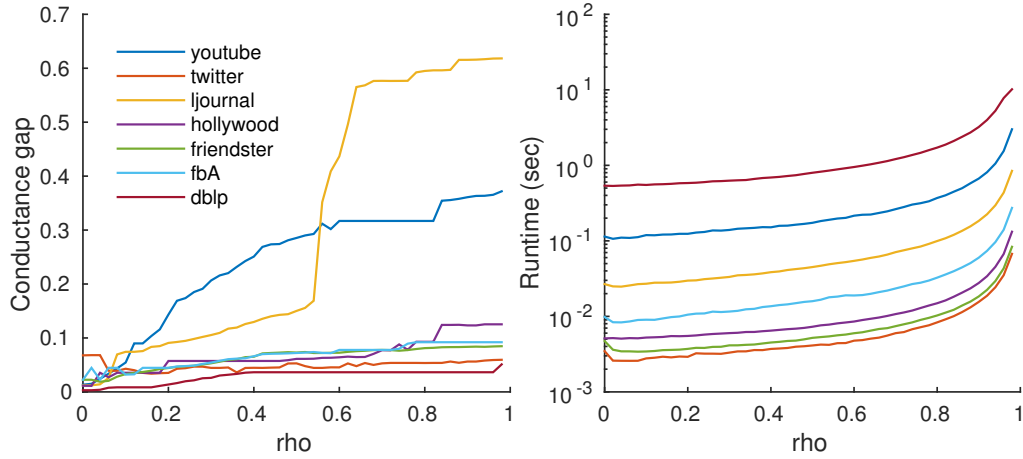


Figure 7. Here we display the behavior of the solution paths as  $\rho$  scales from 0 to 1. At left, we display the gap between  $\phi(\rho)$ , the best conductance found at that value of  $\rho$ , and  $\phi_{\min}$ , the minimum conductance found over all values of  $\rho$ . The lines depict the maximum difference over 100 trials of the quantity  $\phi(\rho) - \phi_{\min}$ . This plot shows that the best conductance found becomes *worse* as  $\rho$  approaches 1. At right, the runtime appears to scale with  $1/(1 - \rho)$ , which is better than the  $1/(1 - \rho)^2$  predicted by our theory.

The greatest difference between the best conductance found for any value of  $\rho$  and the worst conductance found for any  $\rho$  occurs in the livejournal graph, with a gap of nearly 0.7. We discovered that the cause for this disparity is that large values of  $\rho$  delay the propagation of the diffusion, and so the  $\rho = 0.9$  paths at  $\varepsilon = 10^{-5}$  did not spread far enough to find a set of conductance near 0.07. In contrast, all paths with  $\rho < 0.5$  did diffuse deep enough into the graph to identify this good conductance set. Thus, it is possible that many of the differences in conductance performance between paths with different values of  $\rho$  might in fact be caused by the *size* of the region to which the diffusion spreads for a given value of  $\varepsilon$ . Figure 8 illustrates this finding.

Our conclusion from these experiments is that, for the goal of finding sets of small conductance, we should use small values of  $\rho$  near zero. While it sometimes happens that  $\rho > 0$  slightly improves conductance, this is not a reliable observation, and so for the

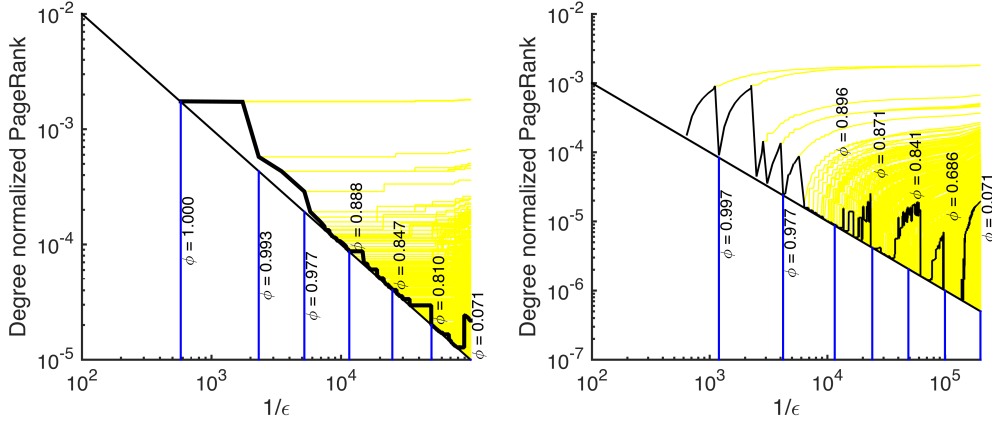


Figure 8. At left, the  $\rho = 0$  paths identify mostly poor conductance sets  $\phi \approx 0.8$ , and locate a set of low conductance,  $\phi = 0.0788$ , only toward the end of the diffusion. At right we see that the  $\rho = 0.9$  paths cannot find this set with  $\varepsilon = 10^{-5}$ . With a slightly smaller accuracy ( $\varepsilon = 5 \cdot 10^{-6}$  instead of  $\varepsilon = 10^{-5}$ ), the diffusion is able to spread far enough to locate the good conductance set.

remaining experiments on conductance, we set  $\rho = 0$ . (This has the helpful side effect of making it easier to compare with our **ppr-grid**.)

## 6.2 Runtime and conductance: ppr-path

Our first method, **ppr-path**, is aimed at studying how PPR diffusions vary with the parameter  $\varepsilon$ . Toward this, Table 2 emphasizes the sheer volume of distinct  $\varepsilon$ -approximations that **ppr-path** explores. We also want to highlight both the efficiency of our method over the naïve approach for computing the solution paths, and the additional information that the solution paths provide compared to a single diffusion.

With this in mind, our experiment proceeds as follows. On each data set, we selected 100 distinct nodes uniformly at random, and ran three personalized PageRank algorithms from that node, with the settings  $\alpha = 0.99$  and  $\varepsilon = 10^{-5}$ . Table 2 displays results for our solution paths algorithm (“path” in the table) compared with two other algorithms chosen to emphasize the runtime and the performance of **ppr-path**.

To show how **ppr-path** scales compared to the runtime of a single diffusion, and to emphasize that the solution paths can locate better conductance sets in some cases, we compare our solution paths method with a standard implementation for computing a single PPR diffusion (“single” in Table 2). Column 3 in the table gives the median runtime, taken over 100 trials, of the single diffusion. To compare, column 4 gives the median *ratio* of “path” time to “single” time. Although **ppr-path** is slower on the small graphs, on the larger graphs we see the runtime is nearly the same as for a single PPR diffusion. At the same time, column 2 shows that “path” computes the results from hundreds or even thousands of diffusions, a significant gain in information over the single PPR diffusion. Finally, column 7 gives the best ratio of conductance found by “path” compared to that

Data	num $\varepsilon$	Single diff. time (sec.)			ppr-path time (sec.)			multi diff. time (sec.)			$\phi$ -ratio
		25	50	75	25	50	75	25	50	75	
itdk0304	5292	0.02	0.02	0.03	0.28	0.41	0.69	70.8	94.2	123.2	1.77
dblp	8138	0.02	0.02	0.02	0.40	0.51	0.65	87.3	97.9	111.5	1.12
youtube	2844	0.01	0.01	0.01	0.05	0.10	0.15	28.6	38.7	49.2	1.47
fb-one	3464	0.01	0.01	0.01	0.03	0.05	0.07	28.1	34.6	40.5	1.09
fbA	862	< 0.01	< 0.01	0.01	0.01	0.01	0.01	14.0	16.5	19.5	1.16
ljournal	2799	0.01	0.01	0.01	0.01	0.02	0.05	24.5	30.9	43.6	2.09
hollywood	423	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	14.0	17.2	22.4	1.19
twitter	172	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	6.5	10.3	18.1	1.05
friendster	402	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	11.1	13.6	16.6	1.09

Table 2. Runtime and conductance comparison of the solution paths (all accuracies from  $10^{-1}$  to  $10^{-5}$ ) with (1) a single PPR diffusion with accuracy  $10^{-5}$  (labelled “single”) and (2) 10,000 PPR diffusions, accuracies  $k^{-1}$  for  $k = 1$  to 10,000 (labelled “mult”). On each dataset we selected 100 distinct nodes uniformly at random and ran the algorithms with the settings  $\alpha = 0.99$  and  $\varepsilon = 10^{-5}$  and  $\rho = 0$ . Column “num  $\varepsilon$ ” displays the median number of distinct accuracy parameters  $\varepsilon$  explored by our algorithm **ppr-path**. Columns under “Time” report 25th, 50th, and 75th percentile of runtimes over these 100 trials. The column “ $\phi$ -ratio” lists the largest (best) ratio of conductance achieved by a single diffusion with conductance achieved by our **ppr-path**, showing our method can improve on the conductance found by a single diffusion by as much as a factor of 2.09.

found by “single”. This shows that the solution paths can improve conductance by 10% to even 50% compared to a single diffusion.

To display the efficiency of our algorithm in computing these many diffusion settings, we again use the standard PPR implementation, but this time set to compute the diffusion for every accuracy setting  $k^{-1}$  for  $k = 1$  to 10,000. This algorithm is “mult” in Table 2, and is essentially a naïve method for approximating the solution paths. Column 5 gives the ratio of “mult” time to “single” time, and shows that this naïve approach to computing diffusions with multiple accuracies is prohibitively slow – it is thousands of times slower than our “path” method.

Lastly, we acknowledge here that both variations on the PPR diffusion are naïve approaches to the problem at hand. However, currently there is no other algorithm for computing the PPR solution paths which we can use as a more competitive baseline.

### 6.3 Runtime and Conductance: ppr-grid

We compare our second method **ppr-grid** with a method called **ppr-grow**, which uses the push framework described in Section 3. Each of these algorithms uses a variety of accuracy settings, and returns the set of best conductance found from performing a sweep-cut over the diffusion vector resulting from each accuracy setting. The algorithm **ppr-grow** has 32 pre-set accuracy parameters  $\varepsilon_k$ . In contrast with **ppr-grid**, which takes its accuracy parameters from a log-spaced grid  $\varepsilon_k = \varepsilon_0 \theta^k$ , the parameters for **ppr-grow** are chosen as the inverses of values from the grid  $10^j \cdot [2 \ 3 \ 4 \ 5 \ 10 \ 15]$  for  $j = 0, 1, \dots, 4$ , along with two additional parameters,  $10^{-6}/2$  and  $10^{-6}/3$ .

In addition to  $\alpha$ , our method **ppr-grid** has the parameters  $\varepsilon_0$  and  $\varepsilon_N$ , the laxest and strictest accuracies (respectively), and  $\theta$ , which determines the fineness of the grid of



accuracy parameters. We use the values  $\varepsilon_0 = 10^{-1}$  and  $\varepsilon_N = 10^{-6}/3$ , and use values of  $\theta$  corresponding to  $N = 32, 64$ , and 1256 different accuracy parameters.

We emphasize that this comparison with the **ppr-grow** method is not as naïve as it might seem: out of the 32 calls that it makes, in practice the very last call (with the strictest value of  $\varepsilon$ ) constitutes near 37% of the total runtime. This means that making only a single call would save little work, and would sacrifice the information from the other 31 (smaller) approximations. Furthermore, the primary optimizations that would be made to the **ppr-grow** framework to improve on this are exactly the optimizations that we make with our **ppr-grid** algorithm, namely avoiding re-doing push work between diffusion computations for different values of  $\varepsilon$ .

Because the two algorithms compute the same PageRank diffusion, comparing their runtimes here allows us to study what proportion of the total work is made up of redundant push operations, and what proportion is comprised of the sweep cut procedures, which both algorithms perform anew for each diffusion. To study this, we highlight the results in Table 3 which displays the runtimes for **ppr-grow** and the *ratios* of the runtimes of **ppr-grid** with **ppr-grow** for computing the best-conductance set from the same number of different diffusions,  $N = 32$ . We also display **ppr-grid** results for the cases  $N = 64$  and 1256 to show how the algorithm scales with the fineness of the grid.

To compare runtimes, we perform the following for each different dataset. For 100 distinct nodes selected uniformly at random, we ran both algorithms with the setting  $\alpha = 0.99$ . We display the best (25%) and worst (75%) quartile of performance of each algorithm and parameter setting. On almost all datasets, we see that **ppr-grid** with  $N = 32$  has a speedup of a factor 2 to 3. This is consistent with our theoretical comparison of the two runtimes in Theorem 5.2, which predicts a factor of 2.9 difference in the push-work that the two algorithms perform. Then, columns 6 through 9 of Table 3 display how quickly **ppr-grid** can compute even more diffusions: whereas **ppr-grow** takes around 1 second to compute and analyze  $N = 32$  diffusions, **ppr-grid** takes little more than half that time to compute on  $N = 64$  diffusions (columns 6 and 7). Columns 8 and 9 show that **ppr-grid** can compute and analyze  $N = 1256$  diffusions, nearly 40 times as many as **ppr-grow**, in an amount of time only 1.10 to 6.59 times greater than the time required by **ppr-grow**.

The conductances displayed in Table 4 are taken from the same trials as the runtime information in Table 3. As with the table of runtimes, for each dataset the table gives the 25% (best) and 75% (worst) percentiles of conductance scores produced by each algorithm on the 100 trials. We see nearly identical conductance scores for **ppr-grow** and **ppr-grid** with  $N = 32$ , which we expect because the two perform nearly identical work. It is interesting to note, however, that increasing the number of diffusions can result in significantly improved conductance scores in some cases, as with  $N = 1256$  on the “fb-one” and “hollywood” datasets. This demonstrates concretely the potential effect of using a broad swath of parameter settings for  $\varepsilon$  to study the meso-scale structure. Moreover, it demonstrates that even a finely spaced mesh of  $\varepsilon$  values, as with **ppr-grow** and **ppr-grid** with  $N = 64$ , can miss informative diffusions.

Data	time (sec.) ppr-grow		time ratio ppr-grid $N = 32$		time ratio ppr-grid $N = 64$		time ratio ppr-grid $N = 1256$	
	25	75	25	75	25	75	25	75
itdk0304	6.23	8.73	0.56	0.61	0.61	0.66	1.10	1.20
dblp	4.52	7.21	0.56	0.62	0.62	0.67	1.28	1.43
youtube	1.73	2.39	0.39	0.50	0.54	0.65	3.35	4.38
fb-one	1.25	1.60	0.33	0.39	0.45	0.53	3.72	4.38
fbA	0.49	0.65	0.47	0.55	0.63	0.72	5.99	6.59
ljournal	0.82	1.20	0.44	0.55	0.58	0.74	4.57	6.12
hollywood	0.28	0.64	0.34	0.49	0.44	0.60	3.47	5.00
twitter	0.13	0.37	0.39	0.44	0.54	0.60	4.61	5.44
friendster	0.34	0.49	0.39	0.44	0.51	0.58	3.90	4.32

Table 3. Runtime comparison of our **ppr-grid** with **ppr-grow**. For each dataset, we selected 100 distinct nodes uniformly at random and ran **ppr-grow** with 32 and **ppr-grid** with  $N$  different accuracy settings  $\varepsilon_k$ . Columns 2 and 3 display the 25th and 75th percentile runtimes for **ppr-grow** (in seconds). The other columns display the median over the 100 trials of the *ratios* of the runtimes of **ppr-grid** (using the indicated parameter setting) with the runtime of **ppr-grow** on the same node. These results demonstrate that our algorithm computing over  $N = 32$  accuracy parameters  $\varepsilon_k$  achieves the factor of 2 to 3 speed-up predicted by our theory in Section 5.3.

Data	grow	$N = 32$		$N = 64$		$N = 1256$	
		25	75	25	75	25	75
itdk0304	0.06	1.00	1.00	1.00	1.01	1.00	1.02
dblp	0.07	1.00	1.00	1.00	1.00	1.00	1.01
youtube	0.18	1.01	1.30	1.09	1.50	1.21	1.72
fb-one	0.37	1.06	1.16	1.10	1.26	1.18	1.37
fbA	0.56	1.00	1.05	1.00	1.06	1.00	1.09
ljournal	0.32	1.00	1.01	1.00	1.01	1.00	1.01
hollywood	0.29	1.00	1.01	1.00	1.01	1.00	1.02
twitter	0.80	1.00	1.00	1.00	1.00	1.00	1.00
friendster	0.85	1.00	1.00	1.00	1.00	1.00	1.01

Table 4. Conductance comparison of our **ppr-grid** with **ppr-grow**. Column 2 displays the median of the conductances found by **ppr-grow** in the same 100 trials presented in Table 3. The other columns display the 25% and 75% percentiles of the *ratio* of the conductances achieved by **ppr-grow** and **ppr-grid** for the same seed set. For example, on the dataset ‘fb-one’, the conductances found by **ppr-grow** are 18% larger than those found by **ppr-grid** with  $N = 1256$  accuracy settings — and that comparison is on the quartile of trials where **ppr-grid** compares the *worst* to **ppr-grow**. We report the ratios in this manner (rather than their reciprocals) because in this form the values displayed are greater than 1, which distinguishes the values from conductance scores (which are between 0 and 1).

## 7 Related work

As we already mentioned, regularization paths are common in statistics [9, 15], and they help guide model selection questions. In terms of clustering and community detection, solution paths are extremely important for a new type of convex clustering objective

function [16, 22]. Here, the solution path is closely related to the number and size of clusters in the model.

One of the features of the solution path that we utilize to understand the behavior of the diffusion is the stability of the set of best conductance over time. In ref. [8], the authors use a closely related concept to study the persistence of communities as a different type of temporal relaxation parameter varies. Again, they use the stability of communities over regions of this parameter space to indicate high-quality clustering solutions.

In terms of PageRank, there is a variety of work that considers the PageRank vector as a function of the teleportation parameter  $\alpha$  [5, 20]. Much of this work seeks to understand the sensitivity of the problem with respect to  $\alpha$ . For instance, we can compute the derivative of the PageRank vector with respect to  $\alpha$ . It is also used to extrapolate solutions to accelerate PageRank methods [6]. More recently, varying  $\alpha$  was used to show a relationship between personalized-PageRank-like vectors and spectral clustering [23]. Note that PageRank solution paths as  $\alpha$  varies would be an equally interesting parameter regime to analyze. The parameter  $\alpha$  functions akin to  $\varepsilon$  in that large values of  $\alpha$  cause the diffusion to propagate further in the graph.

## 8 Conclusions and discussion

We proposed two algorithms that utilize the push step in new ways to generate refined insights on the behavior of diffusions in networks. The first is a method to rapidly estimate the degree-normalized PageRank solution path as a function of the tolerance  $\varepsilon$ . This method is slower than estimating the solution of a single diffusion in absolute run time, but still fast enough for use on large graphs. We designed that method, and the associated degree-normalized PageRank solution path plot, in order to reveal new insights about regions at different size-scales in large networks. The second method is a fast approximation to the solution path on a grid of logarithmically-spaced  $\varepsilon$  values. It uses an interesting application of bucket sort to efficiently manage these diffusions. We demonstrate that both of these algorithms are fast and local on large networks.

The seeded PageRank solution plots, in particular, are effective at identifying a number of subtle structures that emerge as a diffusion propagates from a set of seed nodes to the remainder of the network. We hope that these become useful tools to diagnose and study the properties of large networks.

As recently established by Ghosh et al. [10], there are many related diffusion methods that all share Cheeger-like inequalities for specific definitions of conductance. We anticipate that our solution path algorithm could apply to any of these diffusions as well. For instance, our recent result on estimating the heat kernel diffusion in large graphs is based on the push step as well [18]; we anticipate only mild difficulty in adapting our results to that diffusion.

Fast access to the solution path trajectories provides a number of additional opportunities that we have not yet explored. We may be able to track multiple clusters directly by managing intermediate data. We may be able to find near-optimal conductance sets that are larger than those that directly optimize the objective. Also, nodes in an egonet or larger set could be further clustered by properties of their solution paths instead of their connectivity patterns.

### Acknowledgments

We thank the following people for their careful reading of several early drafts: Huda Nassar, Bryan Rainey, and Varun Vasudevan. This work was supported by NSF CAREER Award CCF-1149756.

### References

- [1] R. Andersen and F. Chung. Detecting sharp drops in pagerank and a simplified local partitioning algorithm. In *Theory and Applications of Models of Computation*, pages 1–12. 2007.
- [2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *FOCS*, 2006.
- [3] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: Model and applications. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM '08*, pages 609–618, New York, NY, USA, 2008. ACM.
- [4] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th WWW2011*, pages 587–596, March 2011.
- [5] P. Boldi, M. Santini, and S. Vigna. PageRank: Functional dependencies. *ACM Trans. Inf. Syst.*, 27(4):1–23, 2009.
- [6] C. Brezinski, M. Redivo-Zaglia, and S. Serra-Capizzano. Extrapolation methods for pagerank computations. *Comptes Rendus Mathématique*, 340(5):393 – 397, March 2005.
- [7] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '09*, pages 219–228, New York, NY, USA, 2009. ACM.
- [8] J.-C. Delvenne, S. N. Yaliraki, and M. Barahona. Stability of graph communities across time scales. *Proceedings of the National Academy of Sciences*, 107(29):12755–12760, June 2010.
- [9] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Ann. Statist.*, 32(2):407–499, 04 2004.
- [10] R. Ghosh, S.-h. Teng, K. Lerman, and X. Yan. The interplay between dynamics and networks: Centrality, communities, and cheeger inequality. In *KDD*, pages 1406–1415, 2014.
- [11] D. F. Gleich. PageRank beyond the web. *SIAM Review*, 57(3):321–363, August 2015.
- [12] D. F. Gleich and M. M. Mahoney. Algorithmic anti-differentiation: A case study with min-cuts, spectral, and flow. In *ICML*, pages 1018–1025, 2014.
- [13] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD*, pages 597–605, Aug. 2012.
- [14] T. Gutierrez-Bunster, U. Stege, A. Thomo, and J. Taylor. How do biological networks differ from social networks? (an experimental study). In *ASONAM*, pages 744–751, 2014.
- [15] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.
- [16] T. Hocking, J.-P. Vert, A. Joulin, and F. R. Bach. Clusterpath: an algorithm for clustering using convex fusion penalties. In *ICML*, pages 745–752, 2011.
- [17] L. G. S. Jeub, P. Balachandran, M. A. Porter, P. J. Mucha, and M. W. Mahoney. Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. *Phys. Rev. E*, 91:012821, Jan 2015.
- [18] K. Kloster and D. F. Gleich. Heat kernel based community detection. In *KDD*, pages 1386–1395, 2014.
- [19] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news

- media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM.
- [20] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.
  - [21] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, September 2009.
  - [22] F. Lindsten, H. Ohlsson, and L. Ljung. Just relax and come clustering! a convexification of k-means clustering. Technical report, Linköpings universitet, 2011.
  - [23] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *Journal of Machine Learning Research*, 13:2339–2365, August 2012.
  - [24] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM.
  - [25] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74(3):036104, September 2006.
  - [26] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.
  - [27] C. (The Cooperative Association for Internet Data Analysis). Network datasets. [http://www.caida.org/tools/measurement/skitter/router\\_topology/](http://www.caida.org/tools/measurement/skitter/router_topology/), 2005. Accessed in 2005.
  - [28] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using seed set expansion. In *CIKM*, pages 2099–2108, 2013.
  - [29] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *EuroSys*, pages 205–218, 2009.
  - [30] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.*, 45(4):43:1–43:35, Aug. 2013.
  - [31] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 745–754, Dec 2012.
  - [32] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems (NIPS)*, volume 16, pages 321–328, 2003.